Práctica: Protegiendo la entrada/salida

Objetivo: Aprender a filtrar correctamente la entrada de datos y codificar la salida

Desarrollo:

1. Siguiendo el modelo MVC visto en clases construir:
   1.1. Una página JSP con un formulario donde se solicite Nombre, mail y edad.

```
<form action="ContactServlet" method="post">
        Name: <input type="text" name="txtNombre" style="width: 100;"></input>
        Mail: <input type="text" name="txtMail"  style="width: 100"></input>
        Age: <input type="text" name="txtEdad"></input>
        <input type="submit" value="Enviar"></input>
</form>
```

   1.2. Un JavaBean para almacenar esa información

```
public class ContactBean {
        private String name, mail;
        private int edad;

        public void setName(String name) { this.name = name;      }

        public String getName() { return  name;    }
…
```

   1.3. Un Servlet que reciba esos tres datos, construya el bean, lo adjunte a la salida y realice un forward a la página resultado

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

        String name = request.getParameter("txtNombre");
        String mail = request.getParameter("txtMail");
        String edad = request.getParameter("txtEdad");

        String address = null;

        ContactBean bean = new ContactBean();
        bean.setName(name);
        bean.setMail(mail);
        bean.setEdad(Integer.parseInt(edad));

        request.setAttribute("contact", bean);
        address = "Exito.jsp";

        RequestDispatcher dispatcher = request.getRequestDispatcher(address);
        dispatcher.forward(request, response);
}
```

   1.4. Un JSP resultado que muestre los datos del bean recibido

```
<h1>Datos de contacto enviados</h1>
<jsp:useBean id="contact" type="javaBeans.ContactBean" scope="request" />
Name: <jsp:getProperty property="name" name="contact" />
…
```

2. Comprobar su funcionamiento

3. Filtrar la entrada, para ello:
   3.1. Crear una clase estática donde se añadirán métodos de filtrado
      3.1.1. Filtro Boolean isEmpty(String in)
      3.1.2. Filtro Boolean isNumber(String in)
      3.1.3. Filtro Boolean isMail(String in)

```
public static Boolean isMail(String in) {

        Pattern p = Pattern.compile(".+@.+\\.[a-z]+");

        Matcher m = p.matcher(in.subSequence(0, in.length()));
        boolean matchFound = m.matches();

        StringTokenizer st = new StringTokenizer(in, ".");
        String lastToken = null;
        while (st.hasMoreTokens()) {
                lastToken = st.nextToken();
        }

        if (matchFound && lastToken.length() >= 2 && in.length() - 1 != lastToken.length()) {
                return true;
        } else
                return false;
}
```

4. Usar los filtros en el Servlet para realizar el forward a otra página en caso de error.
5. Comprobar que funciona adecuadamente

6. Analicemos la salida, introducir XSS en el nombre y comprobar si es posible la inyección.
7. Añadir el siguiente filtro XSS como *Filter* en vuestro proyecto.

```
public class XSSFilter implements Filter {

        FilterConfig filterConfig = null;

        public void destroy() {
                this.filterConfig = null;
        }

        public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws
IOException, ServletException {
                chain.doFilter(new RequestWrapper((HttpServletRequest) request), response);
        }


        public void init(FilterConfig fConfig) throws ServletException {
                this.filterConfig = fConfig;
        }
}
```

```
public class RequestWrapper extends HttpServletRequestWrapper {

	private static final Pattern XSS_PATTERN = Pattern.compile("<([ ])*(?i)script([^>])*>|<([ ])*/([ ])*(?i)script([^>])*>");

	public RequestWrapper(HttpServletRequest servletRequest) {
		super(servletRequest);
	}

	public String[] getParameterValues(String parameter) {

		String[] values = super.getParameterValues(parameter);
		if (values==null)  {
			return null;
		}
		int count = values.length;
		String[] encodedValues = new String[count];
		for (int i = 0; i < count; i++) {
			encodedValues[i] = cleanXSS(values[i]);
		}
		return encodedValues;
	}

	public String getParameter(String parameter) {
		String value = super.getParameter(parameter);
		if (value == null) {
			return null;
		}
		return cleanXSS(value);
	}

	public String getHeader(String name) {
		String value = super.getHeader(name);
		if (value == null)
			return null;
		return cleanXSS(value);

	}

	private String cleanXSS(String value) {
		if (value!=null){
			//return  value.replaceAll("<([ ])*(?i)script([^>])*>", "").replaceAll("<([ ])*/([ ])*(?i)scRipt([^>])*>", "");
			Matcher m = XSS_PATTERN.matcher(value);
			return m.replaceAll("");
		}else
			return null;
	}

}
```

8. Saltaros el filtro AntiXSS
9. Codificar correctamente la salida utilizando JSLT:
   9.1. Añadir a la página de salida el *taglib* adecuado

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

   9.2. Filtrar la salida

```
Name (JSTL): <c:out escapeXml="true" value="${contact.name}" />
```

10. Configura las páginas de errores en el fichero Web.xml